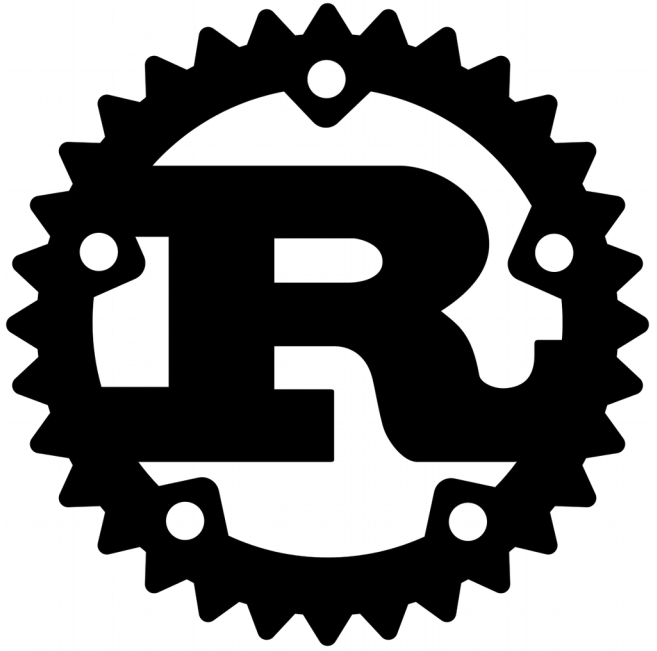

Introducción a RUST

Ricardo Prieto
Twitter: @richiprieto

EÓN CORP

¿RUST?

“Rust es un lenguaje de programación de sistemas estático y fuertemente tipado”

Estático: Todos los tipos son conocidos en tiempo de compilación

Fuertemente: Los tipos están diseñados para hacer más difícil la escritura de programas incorrectos

Sistemas: Va a generar el mejor código de máquina posible con control total del uso de memoria

¿Para qué se lo puede utilizar?

Las aplicaciones pueden ser:

- Desarrollar Sistemas Operativos
- Desarrollo de Drivers
- Programar Sistemas Embebidos
- Desarrollo de Servicios de Red
- Web Assembly

¿Por qué utilizar RUST?

Seguro por default: Todos los accesos de memoria son chequeados y no es posible corromper la memoria por accidente.

Multiparadigma:

- Concurrente
- Funcional
- Genérica
- Imperativa
- Estructurada

¿Debo invertir mi tiempo en RUST?

La curva de aprendizaje es más compleja: Principalmente por que necesitas fundamentos de funcionamiento del procesador, modelo de memoria y concurrencia.

Te hace sentir menos productivo: no puedes ver resultados inmediatamente como con otros lenguajes, pero los códigos iniciales en otros lenguajes seran sub óptimos.

¿Debo invertir mi tiempo en RUST?

Los conceptos pueden ser aprendidos: nunca es una pérdida de tiempo, saber cómo funciona la memoria y la concurrencia, te beneficiará sin importar en qué lenguaje escribas el código.

Fortalecer el cerebro: si no estas aprendiendo algo realmente nuevo todo el tiempo, te estancas y serás una persona con 10 años de experiencia en hacer lo mismo.

Principios unificados RUST

- **Aplicar estrictamente el “safe borrowing” de datos**
- **Funciones, métodos y closures para operar con datos**
- **Tuplas, structs y enums para agregar datos**
- **Pattern matching para seleccionar y desestructurar datos**
- **“Traits” para definir el comportamiento de los datos**

Característica principal en RUST

- **Ownership**
- Todos los datos almacenados en Rust tendrán un propietario asociado
  - Cada dato puede tener un solo propietario a la vez
  - Dos variables no pueden apuntar a la misma ubicación de memoria.
- “Asumiremos que usted acepta que el Garabage Collection (GC) no siempre es una solución óptima, y que es deseable manejar manualmente la memoria en algunos contextos.”

Instalando RUST

- **Linux – Windows Subsystem for Linux**
  - **curl https://sh.rustup.rs -sSf | sh**
- **On-Line**
  - **https://play.rust-lang.org/**

“Hola mundo” RUST

-

```rust
fn main(){
    println!("Hola Mundo");
}
```

“Ownership” RUST

- **Ownership**

```rust
fn main(){
    let a = vec![1, 2, 3];
    let b = a;
    println!("{:?}", b);
    println!("{:?}", a);
}
```

```
println!("{:?}", a);
                 ^ value borrowed here after move
```

“Ownership” RUST

- **No aplica para primitivas, porque necesita menos recursos que un objeto**

```
1  fn main(){
2      let a = 1;
3      let b = a;
4      println!("{}", a);
5      println!("{}", b);
6  }
```

Variables RUST

- **Las variables son por default: Solo Lectura**

```rust
fn main(){
    let a = 32;
    a = a+1;
    println!("{}", a);
}
```

```
let a = 32;
    -
    |
    first assignment to `a`
    help: make this binding mutable: `mut a`
a = a+1;
^^^^^^^ cannot assign twice to immutable variable
```

Variables RUST

- **Usar “mut” para ponerlas en modo escritura**

```
1  fn main(){
2      let mut a = 32;
3      a = a+1;
4      println!("{}", a);
5  }
```

If - for RUST

- **For:**

```rust
fn main() {
    for i in 0..5 {
        if i % 2 == 0 {
            println!("par {}", i);
        } else {
            println!("impar {}", i);
        }
    }
}
```

Funciones - RUST

- **Tipos Explícitos:**

```rust
fn sqr(x: f64) -> f64 {
    return x * x;
}

fn main() {
    let res = sqr(2);
    println!("El cuadrado es {}", res);
}
```

```
let res = sqr(2);
              ^
              |
              expected f64, found integer
              help: use a float literal: `2.0`
```

Funciones - RUST

- **Tipos Explícitos:**

```rust
fn sqr(x: f64) -> f64 {
    return x * x;
}

fn main() {
    let res = sqr(2 as f64);
    println!("El cuadrado es {}", res);
}
```

Arrays y Slices - RUST

- **Arrays son te tamaño fijo y de un solo tipo:**

```rust
fn main() {
    let arr = [10, 20, 30, 40];
    let first = arr[0];
    println!("primero {}", first);

    for i in 0..4 {
        println!("[{}] = {}", i,arr[i]);
    }
    println!("longitud {}", arr.len());
}
```

Arrays y Slices - RUST

- **Slices:**

```rust
fn main() {
    let enteros = [1, 2, 3, 4, 5];
    let slice1 = enteros[0..2];
    let slice2 = enteros[1..];

    println!("ints {:?}", enteros);
    println!("slice1 {:?}", slice1);
    println!("slice2 {:?}", slice2);
}
```

```
let slice1 = enteros[0..2];
    ^^^^^^   ------------- help: consider borrowing here: `&enteros[0..2]`
    |
    doesn't have a size known at compile-time
```

Arrays y Slices - RUST

- **Slices Borrowing:**

```
1  fn main() {
2      let enteros = [1, 2, 3, 4, 5];
3      let slice1 = &enteros[0..2];
4      let slice2 = &enteros[1..];
5
6      println!("ints {:?}", enteros);
7      println!("slice1 {:?}", slice1);
8      println!("slice2 {:?}", slice2);
9  }
```

Vectores - RUST

- **Vectores son de tamaño dinámico:**

```
1  fn main() {
2      let mut v = Vec::new();
3      v.push(10);
4      v.push(20);
5      v.push(30);
6
7      let primero = v[0];
8
9      println!("v es {:?}", v);
10     println!("primero {}", primero);
11 }
```

Iteradores - RUST

- **Iteradores arrays y vectores:**

```
1  fn main() {
2      let arr = [10, 20, 30];
3      for i in arr.iter() {
4          println!("{}", i);
5      }
6  }
```

String - RUST

- **Strings:**

```rust
fn main() {
    let mut s = String::new(); // vacio inicialmente

    s.push('H');
    s.push_str("ello");
    s.push(' ');
    s += "World!"; // Sumamos al String

    s.pop(); // eliminamos el ultimo caracter

    println!("{}", s);
}
```

Tuplas - RUST

- **Tuplas pueden contener diferentes tipos:**

```rust
fn sum_mul(x: f64, y: f64) -> (f64,f64) {
    (x + y, x * y)
}

fn main() {
    let t = sum_mul(2.0,10.0);

    println!("t {:?}", t);

    println!("sum {} mul {}", t.0,t.1);

    let (add,mul) = t;
    println!("sum {} mul {}", add,mul);
}
```

Match - RUST

- **Match:**

```
1  fn main() {
2      let x = 1;
3
4      match x {
5          1 => println!("uno"),
6          2 => println!("dos"),
7          _ => println!("otro numero"),
8      }
9  }
```

Trait - RUST

- **Trait:**

```rust
trait Show {
    fn show(&self) -> String;
}

impl Show for i32 {
    fn show(&self) -> String {
        format!("i32 {}", self)
    }
}

impl Show for f64 {
    fn show(&self) -> String {
        format!("f64 {}", self)
    }
}

fn main() {
    let entero = 42;
    let pi = 3.14;
    let s1 = entero.show();
    let s2 = pi.show();
    println!("{}", s1);
    println!("{}", s2);
}
```

Utilitarios en RUST

- **Cargo**
- Es el administrador de paquetes en Rust, permite compilar, hacer paquetes distribuibles y cargalos a crates.io
- **rustfmt**
- Herramienta para dar formato a Rust de acuerdo a las guias de estilo
- **Clippy**
- Una colección de lints para detectar errores comunes y mejorar el código en Rust

Bibliografía

- https://doc.rust-lang.org/stable/book/
- https://stevedonovan.github.io/rust-gentle-intro/readme.html
- https://thenewstack.io/safer-future-rust/
- https://www.reddit.com/r/rust/comments/4l44z3/why_should_i_use_rust/